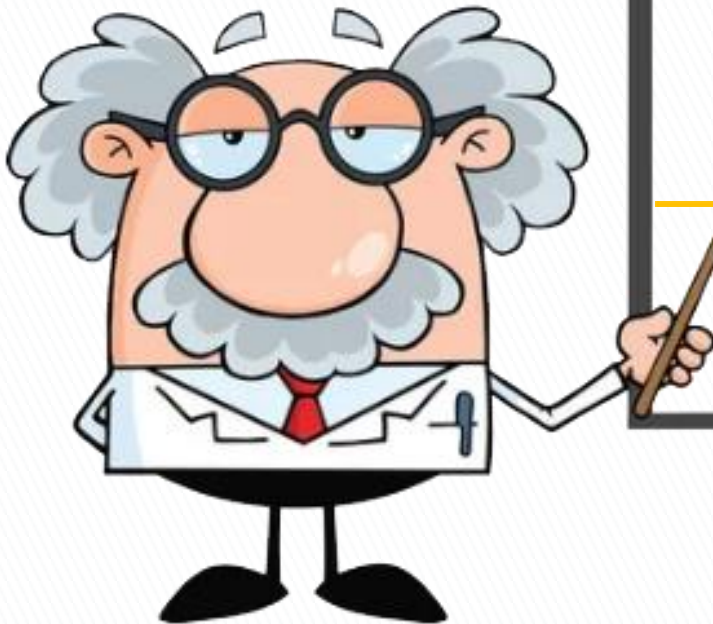




Öğr.Gör. Mustafa GÖKMEN  
gokmen@selcuk.edu.tr



# Nesneye Yönelik Programlama

Döngüler  
(Farklı Kullanım)






# for Döngüsü (Farklı kullanım)


Başlangıç ve kontrol deyimleri birden fazla tanımlanabilir ancak dikkatli bir şekilde kodlanması gereklidir.

```
public class ForLoop02 {  
    public static void main(String[] args) {  
        int i, j;  
  
        for (i = 1, j = 9; ((i <= 9) && (j >= 1)); i++, j--) {  
            System.out.printf("%d + %d = %d\n", i, j, i + j);  
        }  
    }  
}
```



1	+	9	=	10
2	+	8	=	10
3	+	7	=	10
4	+	6	=	10
5	+	5	=	10
6	+	4	=	10
7	+	3	=	10
8	+	2	=	10
9	+	1	=	10

```
public class ForLoop02 {  
    public static void main(String[] args) {  
        int i, j;  
  
        for (i = 1, j = 9; ((i < 9) && (j > 1)); i++, j--) {  
            System.out.printf("%d + %d = %d\n", i, j, i + j);  
        }  
    }  
}
```



1	+	9	=	10
2	+	8	=	10
3	+	7	=	10
4	+	6	=	10
5	+	5	=	10
6	+	4	=	10
7	+	3	=	10
8	+	2	=	10



# Gelişmiş for (foreach) döngüsü

for-each stili bir döngü, dizi gibi bir nesne koleksiyonu üzerinde kesin sıralı bir biçimde baştan sona kadar döngü oluşturmak için tasarlanmıştır. for döngüsünün for-each stili aynı zamanda gelişmiş for döngüsü olarak da bilinir. Genel biçimi aşağıdaki şekildedir.

```
for (tip degisken: Koleksiyon) {  
    degiskeni kullanan kod bloku  
}
```

Burada **tip** baştan sona dek, her seferinde bir tane olmak üzere **Koleksiyon**dan elemanları birer birer alacak olan iterasyona ait değişken tipi ve **degisken** ise bu değişkenin adıdır. Döngü kurulan koleksiyon, **Koleksiyon** ile belirtilir. Döngünün her iterasyonunda, koleksiyonun sıradaki eleman alınır ve **degisken** içinde depolanır. Döngü, koleksiyondaki tüm elemanlar elde edilene kadar tekrarlanır.



# Gelişmiş for (foreach) döngüsü

```
public class ForEach01 {  
    public static void main(String[] args) {  
        Integer[] sayilar = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };  
        for (Integer i : sayilar) {  
            System.out.printf("%2d^2 = %3.0f\n", i, Math.pow(i, 2));  
        }  
    }  
}
```

1^2 = 1  
2^2 = 4  
3^2 = 9  
4^2 = 16  
5^2 = 25  
6^2 = 36  
7^2 = 49  
8^2 = 64  
9^2 = 81  
10^2 = 100

```
public class ForEach02 {  
    public static int maksimum(int[] sayilar) {  
        int enbuyuk = sayilar[0];  
        for (int s : sayilar) {  
            if (s > enbuyuk) {  
                enbuyuk = s;  
            }  
        }  
        return enbuyuk;  
    }  
  
    public static void main(String[] args) {  
        int[] ucet = { 850, 970, 615, 550, 488, 827, 995, 108 };  
        int max = maksimum(ucet);  
        System.out.println("En yüksek ücret: " + max);  
    }  
}
```

En yüksek ücret: 995



# Sonsuz Döngü (Continuous Loop)

`for` ifadesinde başlangıç, koşul ve kontrol deyimleri verilmez ise bu döngü sürekli döngü olarak adlandırılır ve sona erdirmek için kod düzenlenmelidir. Aynı işlem `while(true)` ile de yapılabilir.

```
import java.util.Scanner;

public class ContLoopFor {
    static Scanner kb = new Scanner(System.in);

    public static void main(String[] args) {
        int carpim = 1, sayi = 0;
        for (;;) {
            System.out.print("Sayı? (0=Son): ");
            sayi = kb.nextInt();

            if (sayi == 0) {
                break;
            } else if (sayi == 1) {
                continue;
            }
            carpim *= sayi;
        }
        System.out.println("Çarpım: " + carpim);
    }
}
```

```
import java.util.Scanner;

public class ContLoopWhile {
    static Scanner kb = new Scanner(System.in);

    public static void main(String[] args) {
        int carpim = 1, sayi = 0;
        while (true) {
            System.out.print("Sayı? (0=Son): ");
            sayi = kb.nextInt();

            if (sayi == 0) {
                break;
            } else if (sayi == 1) {
                continue;
            }
            carpim *= sayi;
        }
        System.out.println("Çarpım: " + carpim);
    }
}
```



Öğr.Gör. Mustafa GÖKMEN  
gokmen@selcuk.edu.tr



# Nesneye Yönelik Programlama

UML  
(Unified Modeling Language)







# OOP Dilleri ve Modelleme

- ❖ Gerçek hayattaki problemleri bilgisayarın sanal ortamında çözebilmek için, her şeyden önce problemin uygun şekilde bilgisayar ortamına aktarılması gerekmektedir. Bu işlem "modelleme (modeling)" olarak adlandırılır.
- ❖ Bir sistemle ilgili birden çok model oluşturulabilir. Tek bir model ile sistemin tamamını görmeye çalışmak yerine, üzerinde çalışılan sistemin farklı yönlerini öne çıkaran modeller hazırlanabilir.



Bir UML model örneği



# OOP Dilleri ve Modelleme

- ❖ Bilgisayar ortamında yazılım geliştirebilmek için bir model oluşturulması gerekir. Model gerçek hayattakine ne kadar benzer ya da yakın olursa, program kodlarının da o kadar kolaylaşacağı düşünülür.
- ❖ Modelleme için sırasıyla iki yaklaşım kullanılabilir.
  - Çözümleme
    - ✓ Hedef sistemin ne yapması gerektiğinin belirlendiği aşamadır. Sınıfların özellik ve davranışlarının ne olması gerektiğinin belirlenmesi ile uğraşır.
  - Tasarım
    - ✓ Sınıflar arası ilişkiler ayrıntılı olarak incelenir, gereksinimleri karşılamak üzere ilişkiler, özellikler ve davranışlar şekillendirilir.
- ❖ Tasarım sona erdiğinde, ortaya nesneye yönelik bir model çıkar.



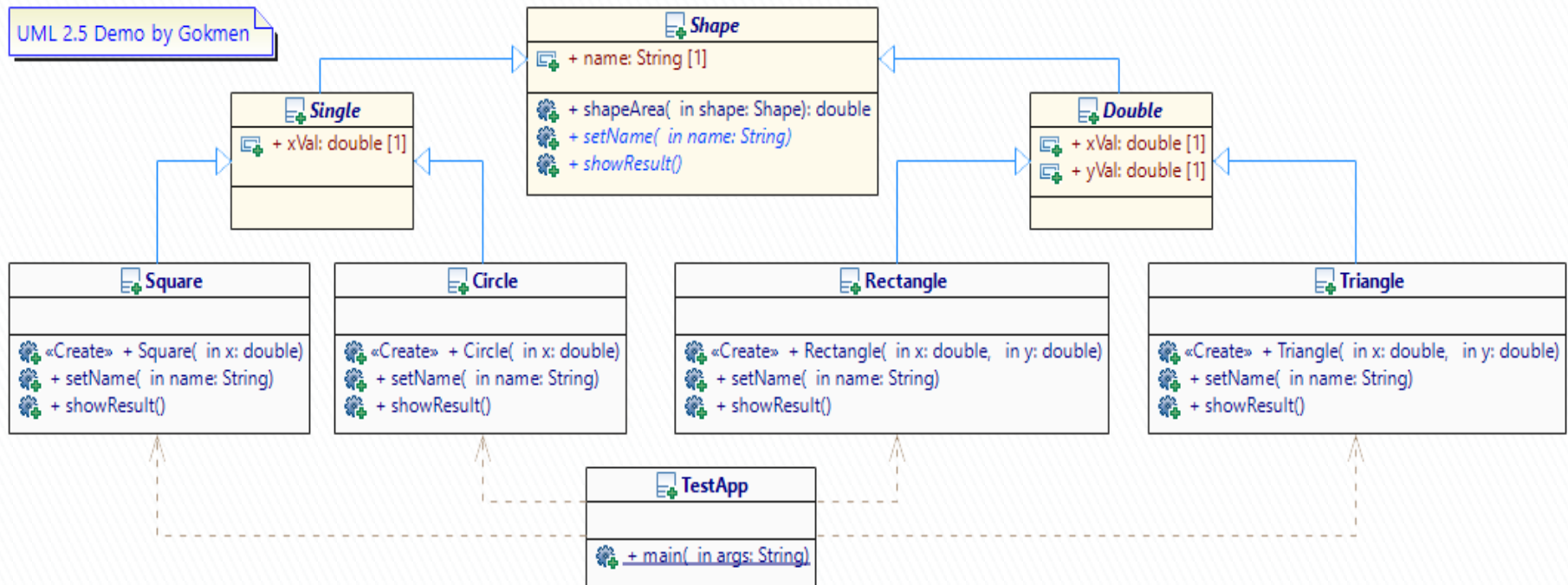


# OOP Dilleri ve Modelleme

- ❖ Modelden nesneye yönelik program yazmaya geçmek, modeli oluşturmaktan daha kolaydır. Çünkü programlama aşaması, modelin bir programlama dili ile ifade edilmesi aşamasıdır ve modeldeki her şeyin programlama dilinde neye karşılık geldiği bellidir.
  - Önemli olan hedef sistemin hangi programlama dili ile geliştirileceği değil, hedef sistem için gerekli olan modelin oluşturulabilmesidir.
- ❖ Sonraki aşamada programlama dili seçimi programlama ekibinin bilgi birikimine, kurumun daha önceki yatırım ve deneyimlerine veya sistemin geliştirilmesini isteyen müşterinin gereksinimlerine göre yapılabilir.



# UML (Unified Modeling Language)



Sistemlerin nasıl modelleneceğini belirleyen ve açıklayan yöntemlerin bir araya toplanmış halidir. UML, yazılımın bileşenlerini ve aralarındaki ilişkileri görsel olarak ifade eden bir standarttır. UML bir programlama dili değildir. UML yazılım geliştirme sürecinin tamamını kapsar.



# UML (Unified Modeling Language)

## *UML diyagram türleri*

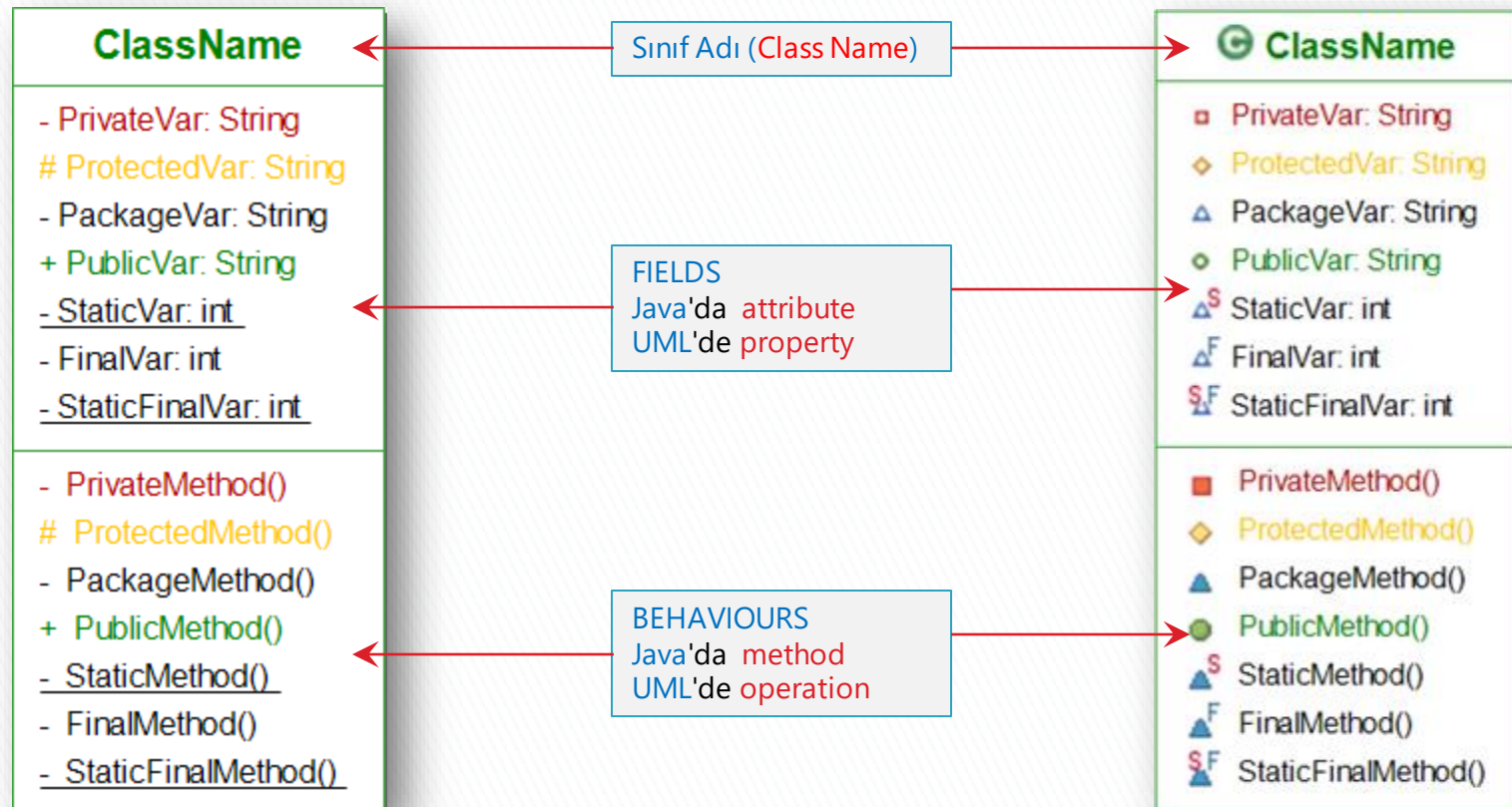
- ❖ Class (Sınıf)
- ❖ Component (Bileşen)
- ❖ Activity (Faaliyet)
- ❖ Use Case (Kullanım Senaryosu)
- ❖ Collaboration (İşbirliği)
- ❖ Sequence (Sıralama)
- ❖ State (Durum)

## *UML kullanım nedenleri:*

- ❖ Yazılımın planlanması görsel olacağından kodlama süresi kısalmış ve olası tasarım hataları önceden belirlenmiş olur.
- ❖ Yazılımcıların aynı temelde iletişim kurmasını sağlar.
- ❖ İleride çıkabilecek geliştirmeleri ve hata düzeltmeleri kolaylaştırır.
- ❖ Projeye sonradan dâhil olan yazılımcıların adapte olmasını kolaylaştırır.
- ❖ Yazılımın mimarisinin daha doğru şekilde oluşturulmasını sağlar.



# UML Diyagram Özellikleri

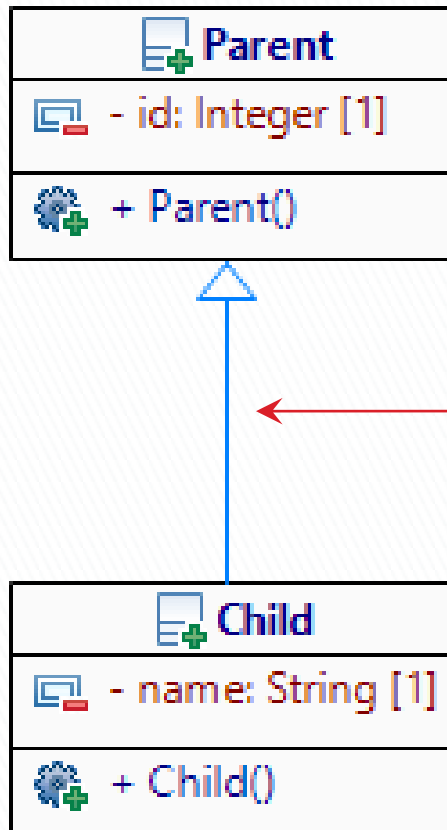






UML (Standard)

UML (Eclipse)



# UML Diyagram Özellikleri



-  Realization (Implementation)
-  Generalization (Inheritance)
-  Dependency (Usage/Import)
-  Association (Relationship)



# UML Diyagram Araçları

## ❖ Visual Paradigm

- ✓ <https://www.visual-paradigm.com/>
- ✓ <https://online.visual-paradigm.com/app/diagrams/>

## ❖ Wondershare eDrawMax

- ✓ <http://www.edrawsoft.com/edraw-uml.html>

## ❖ Diagrams.net

- ✓ <https://www.diagrams.net/>

## Umbrello Project

- ✓ <https://uml.sourceforge.io/>

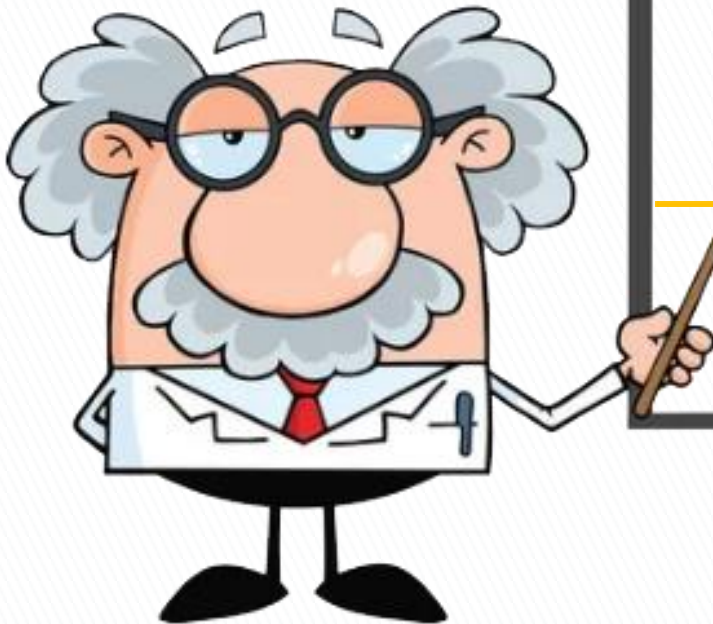
## WhiteStarUML Project

- ✓ <https://whitestaruml.sourceforge.net/>





Öğr.Gör. Mustafa GÖKMEN  
gokmen@selcuk.edu.tr



# Nesneye Yönelik Programlama

OOP Özellikleri  
(Encapsulation)





# Encapsulation

- ❖ Encapsulation (**Kapsülleme**), sınıfın içindeki **metodlar ve değişkenlerin korunmasıdır**. Sınıfın metod ve değişkenlerine dışarıdan erişmenin sakıncalı olduğu durumlarda kapsülleme mekanizması kullanılır.
- ❖ Kapsülleme sayesinde, sınıf üyeleri, dış dünyadan gizlenir. Tabii ki bu gizliliğin dereceleri vardır. Bu dereceler, erişim belirleyiciler (**access modifier**) sayesinde olur.
- ❖ Bu işlem teknik olarak "**Information Hiding**" ya da "**Data Hiding**" olarak bilinir.
- ❖ **Kapsülleme**, sınıf içerisindeki bilgilerin herkes tarafından erişilemez olması, yani **yetkilendirmelere göre erişime açık veya kapalı olması** durumudur.



# Access Modifiers (public access)

**public:** Önüne yazıldığı ögenin kendi sınıfı, türetilen sınıflar, aynı paketteki sınıflar, başka paketteki sınıflar vb. gibi bütün kod kesimlerinden, yani *her yerden doğrudan erişilebilir* olmasını sağlar.

```
// Sınıf bildirimi
public class ClassName {

    // Sınıf Gövdesi

    public int a;

    // Metod bildirimi
    public void myMethod() {
        System.out.println("Public!..");
    }

}
```



# Access Modifiers (protected access)

**protected:** Bu şekilde tanımlanmış bir öge, kendisinin bulunduğu *sınıf ile aynı pakette bulunan sınıflar* ve *bu sınıftan türetilmiş alt sınıflar* içerisinde doğrudan erişilebilir.

```
// Sınıf bildirimi
public class ClassName {

    // Sınıf Gövdesi

    protected int a;

    // Metod bildirimi
    protected void myMethod() {
        System.out.println("Protected!..");
    }

}
```



# Access Modifiers (default access)

**default:** Bir öğenin önüne herhangi bir erişim düzenleyici yazılmazsa, kendi sınıfı ile birlikte *aynı pakette bulunan sınıfların içinden doğrudan erişilebilir*, *başka paketlerden erişilemez*.

```
// Sınıf bildirimi
class ClassName {

    // Sınıf Gövdesi

    int a; // default access (package)

    // Metod bildirimi
    void myMethod() {
        System.out.println("Default!..");
    }
}
```



# Access Modifiers (private access)

**private:** Önüne yazıldığı öğelere, *sadece içerisinde bulunduğu sınıftan doğrudan erişilebilir*. Alt sınıflar da dahil olmak üzere bu *sınıf dışındaki kod kesimlerinden doğrudan erişilemez* olmasını sağlar.

```
// Sınıf bildirimi
public class ClassName {

    // Sınıf Gövdesi

    private int a;

    // Metod bildirimi
    private void myMethod() {
        System.out.println("Private!..");
    }

}
```





# Access Modifiers (private access)

## Getter & Setter Metodları

Bir nesne, sınıfın **private** alanlarına (okuma ve yazma için) erişimde **public** nitelilemeli "**Getter**" & "**Setter**" olarak adlandırılan özel metodlar kullanır.

```
class Ogrenci {  
    private String ogrNo;  
    private String ogrName;  
  
    public void setOgrNo(String ogrNo) {  
        this.ogrNo = ogrNo;  
    }  
    public void setOgrName(String ogrName) {  
        this.ogrName = ogrName;  
    }  
    public String getOgrNo() {  
        return ogrNo;  
    }  
    public String getOgrName() {  
        return ogrName;  
    }  
}
```

```
public class GetterSetter {  
    public static void main(String[] args) {  
        Ogrenci ogr = new Ogrenci();  
  
        ogr.setOgrNo("123");           // Setter  
        ogr.setOgrName("Mustafa GÖKMEN"); // Setter  
  
        System.out.println(ogr.getOgrNo()); // Getter  
        System.out.println(ogr.getOgrName()); // Getter  
    }  
}
```

123  
Mustafa GÖKMEN

- ✓ Getter ve Setter private değere erişiyorlar.
- ✓ Getter ve Setter public erişime sahipler.
- ✓ Getter'in parametresi yok, geri dönüşü var.
- ✓ Setter'in parametresi var, geri dönüşü yok.



# "this" ve "super" keywords

- ❖ Bir sınıfın kendisine ve üst sınıflarına ait varsayılan yapıcılar (constructor) otomatik olarak yürütülür.
- ❖ Aynı sınıfın bir başka yapıcısı `this()` ile ve üst sınıfın bir yapıcısı, alt sınıf yapıcısından `super()` ile çağırılır. İki hususa dikkat edilmesi gerekir:
  - ✓ `this()` ve `super()` her zaman ilk satırda yer alır.
  - ✓ `this()` ve `super()` sadece bir kez kullanılabilir.
- ❖ Herhangi bir yerden `this` ve `super` kelimeleriyle **field** ve **metotlara** erişim için, yukarıda söz edilen kısıtlamalar geçerli değildir.  
Örnek: `this.name;` ve `super.method();` gibi.
- ❖ Bir metod içerisinde, global değişkenler ile aynı adlı yerel değişkenler tanımlanırsa global değişkenler gölgelenmiş (**shadowing**) olur.
- ❖ Bir sınıf/nesne *değişkenine (fields)* erişmek için, `this` anahtar kelimesi kullanılır. (Editörün renklendirmesi de dikkatli gözlerden kaçmamıştır.)

```
int num;
public void add(int num) {
    this.num += num;
}
```

```
class Canli {
    String name;
    Hayvan() {
        name = "Canlı";
    }
    Hayvan(String name) {
        this.name = name;
    }
}

class Kuzu extends Hayvan {
    Kuzu() {
        this("Kuzu");
    }
    Kuzu(String name) {
        this.name = name;
    }
}

class Kurt extends Hayvan {
    Kurt() {
        super("Kurt");
    }
    Kurt(String name) {
        super.name = name;
    }
}
```



# Access Modifiers (Özet)

Erişim Belirleyici	Aynı paket içerisindeki		Farklı paket içerisindeki	
	tüm sınıflardan	alt sınıflardan	tüm sınıflardan	alt sınıflardan
public (+)	✓	✓	✓	✓
protected (#)	✓	✓	✗	✓
default	✓	✓	✗	✗
private (-)	✗	✗	✗	✗

✓ = Erişime açık

✗ = Erişime Kapalı



- **private** bir alan veya metoda sadece aynı sınıftan doğrudan erişmek mümkündür.
- Sınıflar **private** veya **protected** olamazlar.
- Erişim belirleyiciler kendi sınıf öğelerini kısıtlayamazlar.